JMLR: Workshop and Conference Proceedings 36:133–148, 2014 BIGMINE 2014

High density-focused uncertainty sampling for active learning over evolving stream data

Dino Ienco DINO.IENCO@TELEDETECTION.FR
UMR TETIS, Irstea, Montpellier, France
LIRMM, Montpellier, France

Bernhard Pfahringer BERNHARD@CS.WAIKATO.AC.NZ
University of Waikato, Hamilton, New Zealand

Indrė Žliobaitė INDRE.ZLIOBAITE@AALTO.FI
Aalto University, Espoo, Finland
Helsinki Institute for Information Technology, Espoo, Finland

Editors: Wei Fan, Albert Bifet, Qiang Yang and Philip Yu

Abstract

Data labeling is an expensive and time-consuming task, hence carefully choosing which labels to use for training a model is becoming increasingly important. In the active learning setting, a classifier is trained by querying labels from a small representative fraction of data. While many approaches exist for non-streaming scenarios, few works consider the challenges of the data stream setting. We propose a new active learning method for evolving data streams based on a combination of density and prediction uncertainty (DBALStream). Our approach decides to label an instance or not, considering whether it lies in an high density partition of the data space. This allows focusing labelling efforts in the instance space where more data is concentrated; hence, the benefits of learning a more accurate classifier are expected to be higher. Instance density is approximated in an online manner by a sliding window mechanism, a standard technique for data streams. We compare our method with state-of-the-art active learning strategies over benchmark datasets. The experimental analysis demonstrates good predictive performance of the new approach.

Keywords: Active learning, Data streams, Density-based clustering

1. Introduction

Today, more data are being generated continuously than ever before. Streaming data pose serious challenges to data analysis researchers and practitioners. For learning predictive models on streaming data one needs to have continuous access to the true values of the target variable (the true class labels). This labeling phase is usually an expensive and tedious task for human experts. Consider, for example, textual news arriving as a data stream. The goal is to predict if a news item will interesting to given user at a given time. The interests and preferences of the user may change over time. To obtain training data, news items need to be labeled as interesting or not interesting. This requires human labor, which is time consuming and costly. For instance, Amazon Mechanical Turk[1] offers a marketplace for intelligent human labeling.

1. https://www.mturk.com

Labeling can also be costly because it may require expensive, intrusive or destructive laboratory tests. Consider a production process in a chemical plant where the goal is to predict the quality of production output. The relationship between input and output quality might change over time due to constant manual tuning, complementary ingredients or replacement of physical sensors. In order to know the quality of the output (the true label) a laboratory test needs to be performed, which is costly. Under such conditions it may be unreasonable to require true labels for all incoming instances.

A way to alleviate this issue is to query labels for a small representative portion of data. The main challenge is how to select a good subset of instances for learning a model. Such a learning scenario is referred to as *active learning* (Settles, 2010; Fu et al., 2013).

Active learning studies how to label selectively instead of asking for all true labels. It has been extensively studied in pool-based (Lewis and Gale, 1994) and online settings (Cohn et al., 1994). In pool-based settings the decision concerning which instances to label is made by ranking all historical instances (e.g. according to uncertainty) while in online active learning each incoming instance is compared to an uncertainty threshold and the system asks for the true label if the threshold is exceeded. The main difference between online active learning and active learning in data streams is in expectations around changes. In data streams the relationship between the input data and the label may change (concept drift) and these changes can happen anywhere in the instance space while online learning assumes a stationary relationship between examples and their labels. In order to capture concept drifts, which can happen anywhere in the data, it is important to characterize and understand how the examples are related to each other in the data space.

As a recent work in the pool-based settings suggests (Fu et al., 2013), considering the density around instances can improve the results. In the more dense regions in the instance space the benefits of updating the classifier with new data are expected to be larger, because it is likely to improve future classification accuracy for more instances.

The concept of density around instance is usually exploited in unsupervised learning (such as clustering (Rodriguez and Laio, 2014; Tomasev et al., 2014)) where local density of a point can be used to initialize cluster centroids. Local density can be directly estimate as the number of points closer to the considered instance (Rodriguez and Laio, 2014) or derived indirectly (Tomasev et al., 2014) as the number of times the instance appears as a neighbor of other example. These approaches widely demonstrate their efficacy for clustering tasks.

Unfortunately, very few works for active learning in data stream integrate the density (Ienco et al., 2013) in order to choose suitable queries instance. Motivated by this fact we introduce a new density-focused uncertainty sampling method that uses an online density approximation to guide the selection of labeling candidates in the data stream. Our approach works in a fully incremental way: in order to estimate the density of the neighborhood of an incoming instance, we use a sliding window mechanism to compare it to a recent portion of the historical data.

The remainder of this paper is organized as follows. Section 2 overviews existing active learning approaches for data streams and makes connections with semi-supervised learning in data streams. The proposed methodology is presented in Section 3. In Section 4 we present experimental results for a number of real world datasets and we also supply a sensitivity analysis of the sliding window size. Finally, Section 5 concludes the study.

2. Related Work

Online active learning has been the subject of a number of studies, where the data distribution is assumed to be static (Helmbold and Panizza, 1997; Sculley, 2007; Cohn et al., 1994; Attenberg and Provost, 2011). The goal is to learn an accurate model incrementally, without assuming to have all training instances available at the beginning, with minimum labeling effort. One example of such a scenario is malicious URL detection where training data are arrive sequentially Zhao and Hoi (2013). In contrast, in the evolving data streams setting, which is the subject of our study, the goal is to continuously update a model over time so that accuracy is maintained as data distribution is changing. The problem of label availability in evolving data streams has been the subject of several recent studies (Klinkenberg, 2001; Widyantoro and Yen, 2005; Masud et al., 2011; Fan et al., 2004; Huang and Dong, 2007; Zliobaite et al., 2014; Ienco et al., 2013) that fall into three main groups.

The first group of works uses semi-supervised learning approaches to label some of the unlabeled data automatically (Klinkenberg, 2001; Widyantoro and Yen, 2005; Masud et al., 2011), which can only work under the assumption that the class conditional distribution does not change, or, in other words, they assume that their is no concept drift. Semi-supervised learning approaches are conceptually different from the active learning approaches, that are the subject of our study, since the former can only handle changes in the input data distribution, changes in the relation between the input data and the target label cannot be spotted without querying an external oracle as is done in active learning.

The second group of works process data in batches implicitly or explicitly assuming that data is stationary within batches (Lindstrom et al., 2010; Masud et al., 2010; Fan et al., 2004; Huang and Dong, 2007). Such approaches require an external mechanism to handle concept drift. Lindstrom et al. (2010) use uncertainty sampling to label the most representative instances within each new batch. They do not explicitly detect changes, instead they use a sliding window approach, which discards the oldest instances. Masud et al. Masud et al. (2010) use uncertainty sampling within a batch to request labels. In addition, they use the unlabeled instances with their predicted labels for training, making it also another semi-supervised learning approach. A few works integrate active learning and change detection (Fan et al., 2004; Huang and Dong, 2007) in the sense that they first detect change and only if change is detected do they ask for representative true labels using offline active learning strategies designed for stationary data. In this scenario drift handling and active learning can be considered as two mechanisms operating in parallel, but doing so independently. This is the main difference between this scenario and the last one, which combines the two mechanisms more closely together.

Finally, the third group of works use randomization to capture possible changes in the class conditional distribution (Zhu et al., 2007; Zliobaite et al., 2014; Cesa-Bianchi et al., 2006). Cesa-Bianchi et al. (2006) develop an online active learning method for a perceptron based on selective sampling using a variable labeling threshold $b/(b+|p|)$, where b is a parameter and p is the prediction of the perceptron. The threshold itself is based on certainty expectations, while the labels are queried at random. This mechanism could allow adaptation to changes, although they did not explicitly consider concept drift.

Differently from previous works, and in the same direction of our idea, Ienco et al. (2013) proposes a clustering-based active learning method for data streams. In that work density is

captured by means of clustering in a batch-incremental scenario. The clustering step allows to perform stratified sampling considering dense areas to selecting examples for labelling.

3. Method

In this section we describe our new approach DBALStream (**D**ensity **B**ased **A**ctive **L**earning for Data **Stream**s). We work in a fully incremental scenario in which each instance is processed as soon as it arrives. We can define a data stream as $S := \{x_1, x_2, ...x_n, ...\}$ where each x_i is a new instance arriving at time i. This scenario is general enough to model arbitrary real-world data streams. Given a data stream S and a budget b we want to learn a classifier cl with only b of the instances in the stream. How to select the instances to query is challenging for any active learning strategy.

The proposed strategy aims at modelling density around an instance, and is guided by the following hypothesis: an instance is more important to label if it lies in a dense area. For example, suppose we are classifying the sentiments towards news items, arriving online. There are a lot of incoming news about the political situation in Ukraine, and very few news items about research in Antarctica. Suppose, sentiments towards Antarctica news are currently more uncertain. We can label the Antarctica news item and learn to classify in this context very accurately, but if similar news items arrive very rarely in the future, we have little gain. Perhaps, we would better label news related to Ukraine, which are frequent (instance space of high density). This way, assuming that there is no sudden change in density, we can expect improvement in future classification accuracy on more instances.

In order to implement this intuition, given a data point x_i we model its density as the number of times x_i is the nearest neighbor of other instances. We use this value as an estimate of the density around a particular instance and in particular, we use this measure to understand if an instance x_i lies in a dense area, or not.

In a batch scenario this operation can be performed over the whole dataset in order to obtain a good picture of the global behavior.

In our setting, as we deal with data streams, it is infeasible to store all data and perform such an operation on the whole data stream. Therefore, we only consider a window, or buffer, of previously processed examples in order to estimate a local density factor for a new incoming example. Given a window W of previous examples and a data structure $MinDist$ that stores the minimum measured distance for each of the instances in W, the local density factor (ldf) of a new instance x_i is defined as follows:

$$ldf(x_i) = \sum_{x_j \in W} \mathbb{I}\{MinDist(x_j) > dist(x_i, x_j)\} \quad (1)$$

where $\mathbb{I}$ is an indicator function that returns 1 if the condition is true, and 0 otherwise. The function $dist(\cdot, \cdot)$ is a distance function defined over two examples x_i and x_j of the stream. Procedurally speaking, the $MinDist$ data structure is updated each time the indicator function is verified. This means that if $MinDist(x_j) > dist(x_i, x_j)$ then $MinDist(x_j)$ is updated with the value $dist(x_i, x_j)$. The minimum distance computed between x_i and each element of W is used as initial value for $MinDist(x_i)$. Another important aspect to underline is that both W and $MinDist$ cannot exceed a predefined size and for this reason

they work as a queue data structure (First In First Out): when the limit is reached the first instance, the oldest one, in the queue is removed and the new one is pushed at the end.

The local density factor of an instance can supply information about its importance, but it is not enough to understand if an example might be useful or not to label. As shown in (Fu et al., 2013), an important factor to consider is the uncertainty related to an instance that is usually employed in general active learning approaches for data streams (Zliobaite et al., 2014).

In order to combine both these aspects, local density factor and uncertainty of an instance, we build our new proposal extending one of the frameworks proposed in (Zliobaite et al., 2014). We extend the framework used for the *Uncertainty Strategy with Randomization*. This method relies on a randomized dynamic threshold, trying to label the least certain instances within a time interval. This threshold adapts, depending on the incoming data, to align with the budget, and it is also adjusted by a normal random factor. If a classifier becomes more certain, the threshold is increased, so that only the most uncertain instances are captured for labelling, otherwise the threshold is decreased to extract more information for improving the classifier.

The original framework employs the maximum a posteriori probability of the classifier in order to quantify the uncertainty of an instance. This method prefers instances on which the current classifier is less confidence (Fu et al., 2013) and can be formally defined as follow:

$$Confidence(x_i) = max_{cl} P_L(y_{cl}|x_i)$$

A different approach to model the uncertainty of the instance x_i is to consider not only the maximum a posteriori probability, but considering also the second most probable class label. Thus this approach is Margin-based (Fu et al., 2013) and it is prone to select instances with minimum margin between posterior probabilities of the two most likely class labels. The margin is defined as:

$$Margin(x_i) = P_L(y_{cl_1}|x_i) - P_L(y_{cl_2}|x_i)$$

where the cl_1 and cl_2 are respectively the class with the maximum a posteriori probability and the second most probable class.

Our proposal is depicted in Algorithm 1. The algorithm takes as input the instance to evaluate x_t, the stream classifier L, the budget percentage b, the adjusting step s, the threshold θ, the window of previous collected instances W and the $MinDist$ data structure that stores the minimum distance for each element in W. The algorithm first computes the number of times the instance x_t is the nearest neighbor of an instance in W. This operation is performed using Formula 1 and implemented by the function $compute\#timesNN(x_t, W, MinDist)$. This procedure returns an integer representing the local density around example x_t and, at the same time, it updates W and $MinDist$ according to their maximum size. Next, the procedure tests two conditions: the first one ($u/t < b$) verifies the budget constraint, which disallow any new labelling when the budget is already exceeded; the second condition ($lrd(x_i) \neq 0$) takes into account the local density function. In particular this test does not allow to ask for labels for instances that lie outside any local dense areas of the data space. If either of the conditions fails, then the algorithm returns false, otherwise the margin for the instance x_t is computed.

Line 4 to 12 are inspired by the approach proposed in (Zliobaite et al., 2014) where the randomized adaptive threshold method was firstly introduced. In Line 4 the threshold θ is randomized employing a variable ϵ sampled from a Normal distribution $N(0,1)$. This randomization step allows to occasionally label instances that are far from the decision boundary. Line 5 checks the uncertainty of x_t w.r.t. the randomized threshold θ_{ran}. If the test is positive, then the instance will be used to train the classifier and the active learning strategy consumes some budget (Line 6). The threshold θ is updated, considering the adjusting step (Line 7) and the procedure terminates. If the test (Line 5) fails, the Algorithm 1 returns false, no budget is spent and the threshold θ is relaxed in order to increase the chance of labeling the next time.

The DBALStream algorithm is employed in the general data stream classification framework proposed in Section 3.1. In particular, it is used to decide if the true label y_t for the instance x_t should be asked for, or not.

Regarding the computational complexity of our method, the most time consuming steps are the computation of $ldf(x_i)$ and the updating of $MinDist$ structure. Both operations have linear complexity in the size of $|W|$ and they can be done at the same time (we need to scan once the window W for each incoming instance x_i).

Algorithm 1 DBALStream(x_t, L, b, s, u, θ, W, $MinDist$)

Require: x_t: the new instance in the data stream, L: learned classifier, b: budget, s: adjusting step, u: number of instances labeled till now, θ: the actual threshold value, W: set of previous instances used to computed the local density function, $MinDist$: minimum distances for each instance in the window W

1: $ldf(x_t) = compute\#timesNN(x_t, W, MinDist)$
2: **if** $(u/t < b$ and $ldf(x_t) \neq 0)$ **then**
3: $\quad margin(x_t) = P_L(y_{cl1}|x_t) - P_L(y_{cl2}|x_t)$
4: $\quad \theta_{ran} = \theta \times \epsilon$ where $\epsilon \in N(0,1)$
5: $\quad$ **if** $margin(x_t) < \theta_{ran}$ **then**
6: $\quad\quad u = u + 1$
7: $\quad\quad \theta = \theta * (1 - s)$
8: $\quad\quad$ **return true**
9: $\quad$ **else**
10: $\quad\quad \theta = \theta * (1 + s)$
11: $\quad\quad$ **return false**
12: $\quad$ **end if**
13: **end if**
14: **return false**

3.1. General Active Learning Classification Framework

Algorithm 2 presents the general active learning classification framework. It requires as input the data stream S and the budget percentage b. At the beginning (line 1 and 2) the framework initializes the different parameters. In particular we set the θ threshold equal to 1 and the step option s to 0.01. The step option s is used for increasing or decreasing the threshold in the DBALStream procedure.

We use DDDM change detection technique (Gama et al., 2004): the accuracy of the classifier is monitored during the streaming process. If the accuracy starts to decrease (*change warning is signaled* line 6-8) a new background classifier L_n is generated.

At line 9-11 both classifiers are trained in parallel with the incoming instance x_t. If a *change is detected*, the classifier L is replaced by L_n (line 13-15).

Classifier performances are evaluated by means of the prequential schema. The evaluation through the prequential setting involves two steps: i) classify an instance, and ii) use the same instance to train the learner. To implement this strategy, each new incoming instance x_t is first classified using L and only after this step the approach decides if a label is waned for training, or not. The accuracy is computed over the classification results of L. This process continues until the end of the data stream is reached.

Algorithm 2 Active Learning Classification Framework

Require: b: labeling budget
Require: S the data stream
1: $\theta = 1$, $u = 0$, $s = 0.01$, $W = \emptyset$, $MinDist = \emptyset$
2: L =classifier, L_n =classifier
3: **for** $x_t \in S$ **do**
4: **if** (DBALSTREAM(x_t, L, b, s, u, θ, W, $MinDist$)) **then**
5: request the true label y_t of instance x_i
6: **if** change warning is signaled **then**
7: start a new classifier L_n
8: **end if**
9: train classifier L with (x_t,y_t)
10: **if** L_n exists **then**
11: train classifier L_n with (x_t,y_t)
12: **end if**
13: **if** change is detected **then**
14: replace classifier L with L_n
15: **end if**
16: **end if**
17: **end for**

4. Experiments

In this section we evaluate the performance and the quality of the proposed method DBALSTREAM. We compare our algorithm with state of the art methods that are explicitly designed for active learning over data streams. We use the prequential evaluation procedure: each time an instance arrives, we first test the algorithm on it, and then we decide on whether to pay the cost for labeling it and subsequently use it as an input for updating the classifier. In order to evaluate the performance of the different strategies we employ classification accuracy. We use two methods proposed by (Zliobaite et al., 2014): *Random* and *Rand Unc*. The first one is *Random*. It randomly chooses examples for labeling. The second one (*Rand Unc*), proposed in (Zliobaite et al., 2014), uses a randomized variable uncertainty strategy that combines the randomization with maximum a posteriori probability and an adaptive method to avoid consuming too much of the budget when a consecutive run of easy instances is encountered. The last competitor *ACLStream* is a clustering-based active learning approach proposed by Ienco et al. (2013). This approach works in a batch-incremental scenario. It performs stratified sampling, where at first a clustering solution over a batch of data is obtained, and then instances are chosen for labelling considering a combination of their own uncertainty and the uncertainty of the cluster they belong to.

For all methods a warm-up period is introduced. The first 500 instances of each dataset are all labeled and used to train the initial model used by the specific approach. Evaluation using active learning only starts after this warm-up step.

All the methods need an internal classification algorithm to perform the classification and to produce the maximum a posteriori probability. For this reason for all the approaches we use the classifier proposed in (Gama et al., 2004). This classifier is able to adapt itself to drift situations: when the accuracy of the classifier begins to decrease, a new classifier is built and trained with new incoming instances. Considering the batch size, for *ACLStream*, following the original paper, we use a window size of 100 instances and the number of clusters is set to 5. As all the approaches are nondeterministic, each result, for each of the strategies, is averaged over 30 runs.

All the experiments are performed using the MOA data stream software suite (Bifet et al., 2010). MOA is an open source software framework implemented in Java designed specifically for data stream mining.

4.1. Datasets

To evaluate all the algorithms we use four real world benchmark datasets: *Electricity*, *CoverType*, *Airlines*, *KDD99*. *Electricity* data (Harries et al., 1998) is a popular benchmark in evaluating streaming classifiers. The task is to predict the rise or fall of electricity prices (demand) in New South Wales (Australia), given recent consumption and prices in the same and neighboring regions. *Cover Type* data (Bache and Lichman, 2013) is also often used as a benchmark for evaluating stream classifiers. The task is to predict forest cover types or types of vegetation from cartographic variables. Inspired by (Ikonomovska et al., 2010) an *Airlines* dataset was constructed using the raw data from US flight control. The task is to predict whether a given flight will be delayed, given the information of the scheduled departure. The last dataset, *KDD99*, is commonly used as a benchmark anomaly detection task but recently it has also been employed as a dataset for testing data stream algorithms (Masud et al., 2011). One of the big problems with this dataset is the big amount of redundancy among instances. To solve this problem we use the cleaned version named NSL-KDD[2]. To build the final dataset we join both training and test data. Table 1 presents summary characteristics of the datasets. This collection includes both binary and multi-class classification problems, datasets with different numbers of instances (varying between 42k to 829k) and different numbers of features (from 7 to 54). Even though *CoverType*, *KDD99* have no time order variable, we assume that they are presented in time order.

In order to better characterize this collection of datasets, we show for each of them, how the class variable evolves over time. For this purpose, we divided each dataset in batches and for each batch we plot its class values distribution. As the different datasets have different sizes, we choose an adapted batch size to allow for a clear trend visualization. This information can be useful to understand which dataset has dramatic (resp. smooth) changes considering the target variable. Logically, quick changes in the target variable requires more flexible classifiers than stationary situations. The characterization of the datasets is presented in Figure 1. The X axis represents the batch number while the Y axis corresponds to the proportion (as ratio) of instances belonging to a class value. At

2. http://nsl.cs.unb.ca/NSL-KDD/

Dataset	n. of Instances	n. of Features	n. of Classes
Airlines	539 383	7	2
Electricity	45 312	8	2
Cover Type	581 012	54	7
KDD99	148 517	41	2

Table 1: Dataset characteristics

each moment the sum of the different class labels is equal to 1. We can observe different behaviors as a function of the target variable. Some datasets show high fluctuation of different classes over the stream (*Cover Type* and *Electricity*) while other ones show more smooth (or stable) behavior (*airlines* and *KDD99*).

We think that this final set of four datasets is a good benchmark for evaluating the performance of our approach, DBALSTREAM, w.r.t. state of the art methods.

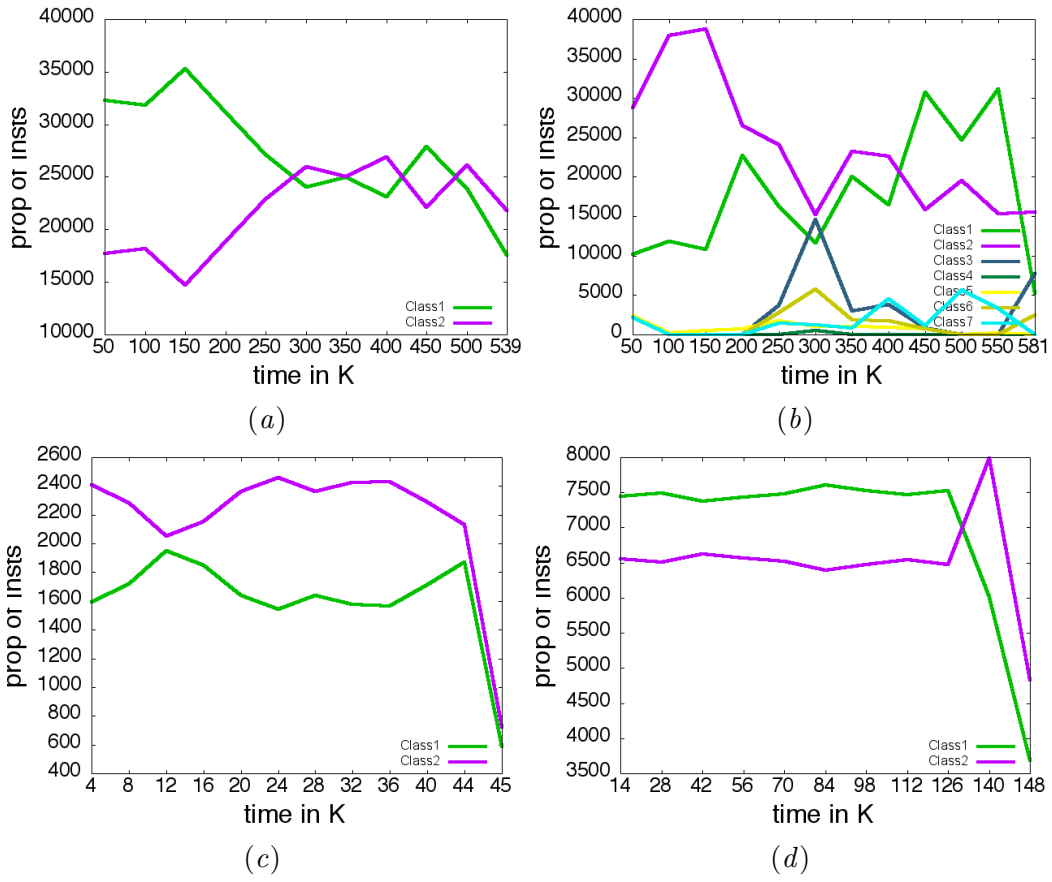

Figure 1: Distribution of classes during the stream process: a) *Airlines* b) *Cover Type* c) *Electricity* e) *KDD99*

4.2. Analysis of Classification Performance

The final accuracy is reported in Figure 2. In this experiment we evaluate the different methods, over the different datasets, varying the budget percentage. We start with a budget percentage of 0.03 and go up to a percentage of 0.3. Obviously, to evaluate the results we need to take into account how the performances change when varying the budget size.

We can observe that DBALStream outperforms the competitors over *Airlines* and *KDD99* datasets while it obtains comparable accuracy results for the other two datasets. In particular, regarding *Cover Type* we can note that all the methods have very similar behavior for budget values smaller or equal to 0.15 while for higher budget the performance of *ACLStream* drops down w.r.t. the competitors. For the *Cover Type* dataset the best result is obtained by DBALStream with a budget value of 0.25. If we analyze the results over *Electricity* we can note that DBALStream has better performance than the other strategies for a big range of the budget value (between 0.1 and 0.25) while for smaller budgets *Rand Unc* slightly outperforms our proposal.

To wrap up this set of experiments we can state that our new proposal obtains better or comparable results with respect to previous approaches in the field of active learning in data streams. These results underline that using the local density information of incoming instances positively affects the active learning process in the data stream setting.

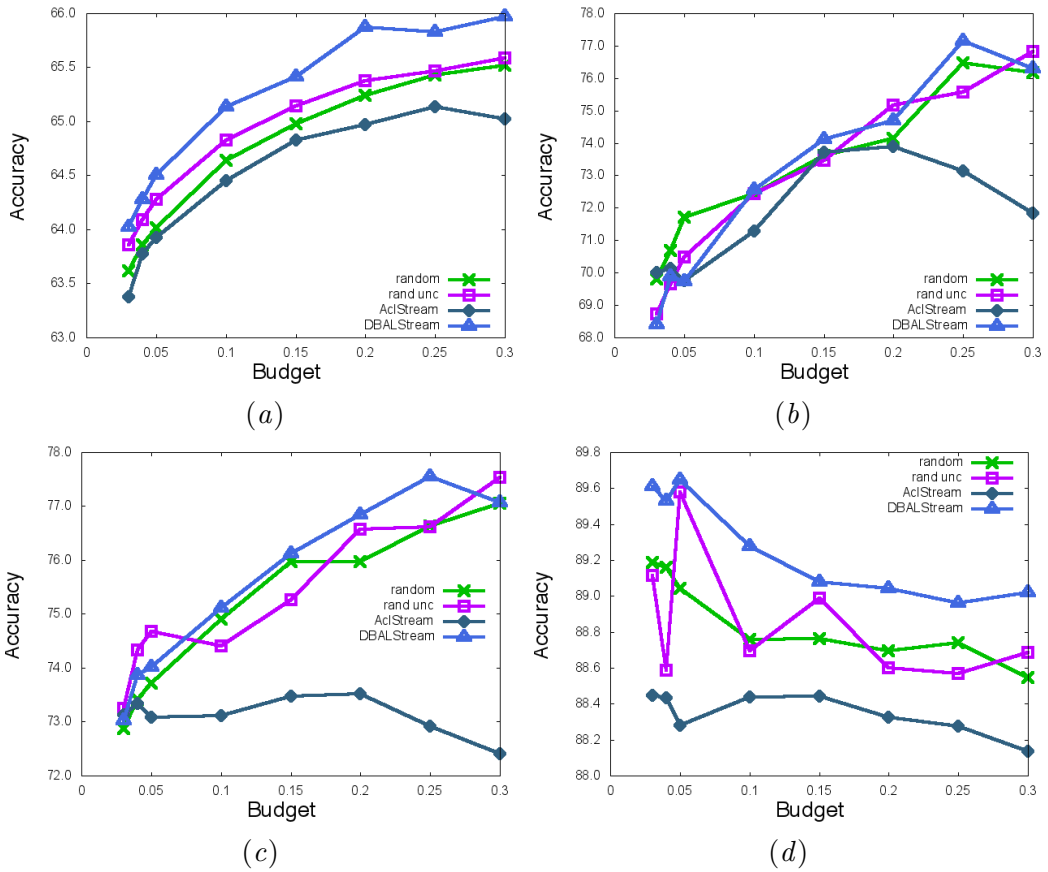

Figure 2: Accuracy on a) Airlines b) Cover Type c) Electricity and e) KDD99

4.3. Influence of the Window Size

In this subsection we focus our attention on the impact of window size over the final performance. In particular, we analyse how the size of the window impacts the final accuracy of DBALSTREAM. For this purpose, we run experiments varying the size of the batches from 50 to 300 with a step of 50. We average each result over 30 runs.

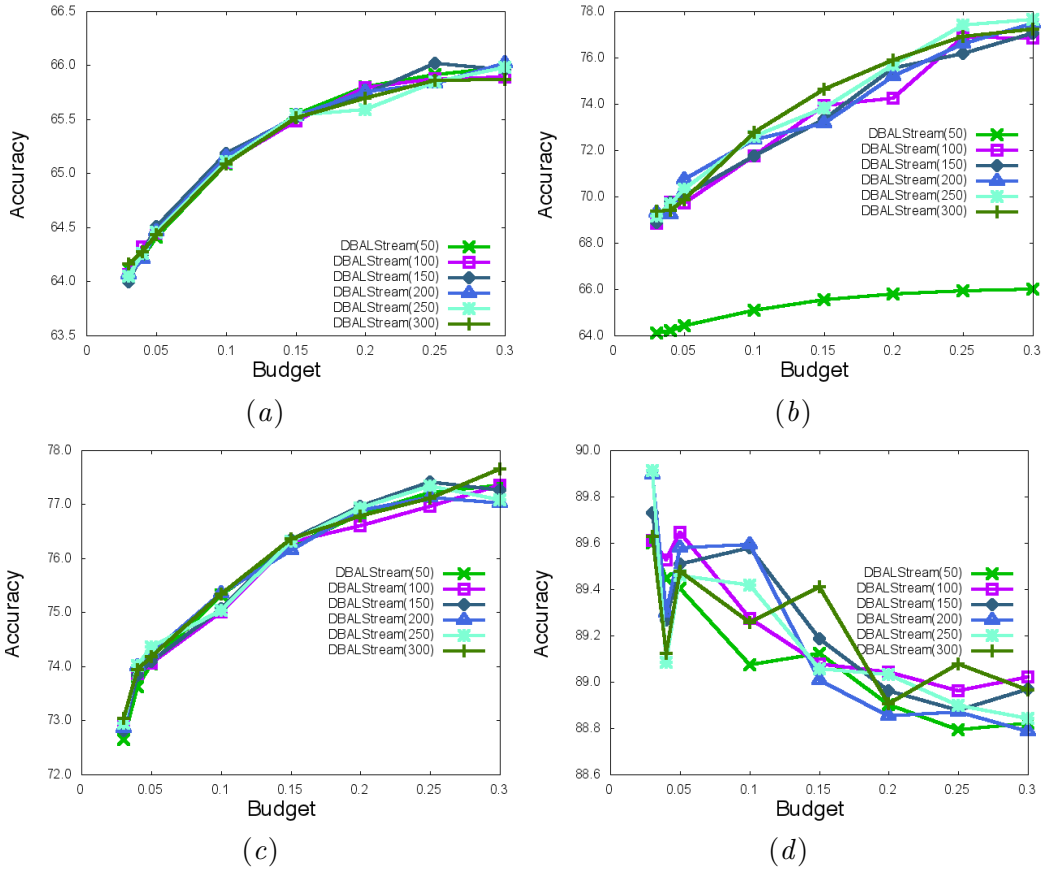

Figure 3: Accuracy Results for DBALSTREAM on a) *Airlines* b) *Cover Type* c) *Electricity* and d) *KDD99* varying the batch size from 50 to 300

The accuracy results are reported in Figure 3. We can observe that, considering *Airlines* and *Electricity* benchmarks the performances of DBALSTREAM are not influenced by this parameter and changes in the value of window size did not impact the final results. A different behavior is shown for the *Cover Type* dataset. In this case we can note an important difference in the results between the smallest value of window size (50) w.r.t. the other values of window size (100-300). This is due to the fact that only considering the 50 previous examples to evaluate the importance of a new one (considering its relative density) is not enough. This can also be explained by the way the data were collected. As we will underline in Section 4.5, the data came from a GIS database that stores raster cells considering their spatial order. These cells are arranged in a matrix and, probably, a window size of 50 did

not allow to capture temporal correlations among the data. For the *KDD99* dataset we note that no particular value of window size outperforms the others. Accuracy fluctuation, also in this case, are very small and they did not exceed 0.5 accuracy points.

In summary, we can underline that for almost all of the benchmarks, this parameter did not affect the final results and window size values bigger than 100 are a good choice in order to compute good density estimates. As a conclusion we can state that, generally, our method is quiet robust with respect to its parameter setting.

4.4. DBALStream: *Confidence* based vs *Margin* based uncertainty estimation

In Section 3 we underlined that our proposal quantifies the uncertainty over an instance considering the *Margin* of the classifier while previous techniques for active learning in data stream exploit the *Confidence* value (Zliobaite et al., 2014). In this section we evaluate how the performances of DBALStream are influenced considering these two different ways of estimating the uncertainty of a classifier over an instance. Figure 4 shows the results of this experiment over all benchmarks datasets.

We can note that, generally, the *Margin* based version clearly outperforms the *Confidence* based one for almost all the datasets and all the budget values. This phenomena is usually more visible for high budget values.

As a general finding we can state that considering the discrepancy between the two most probable classes supplies much more information on the uncertainty of an instance than only taking into account the maximum a posteriori probability. This also supports our choice of using a *Margin*-based approach in our framework.

4.5. Performance with different types of concept drift

For analysis purposes, we also introduce two more datasets: *Cover Type Sorted*, in which the instances of the *Cover Type* dataset are reordered w.r.t. the attribute *elevation*, and *Cover Type Shuffled*, where the order of instances is randomly shuffled. Due to the nature of the underlying problem, sorting the instances by the *elevation* attribute induces a natural gradual drift on the class distribution, because at higher elevation some types of vegetation disappear while other types of vegetation appear gracefully. Shuffling instances ensures that the data distribution is uniform over time, i.e. there is no drift at all. Figure 5 plots the prior probabilities of the classes over time in the two new versions of the dataset

Note, that the order of the *Cover Type* dataset is related to the spatial location of the instances, as clarified by the authors in personal communication. This additional knowledge about the data indicates that in the first two cases (*Cover Type* and *Cover Type Sorted*) the data contains some kind of spatial auto-correlation while in the last scenario (*Cover Type Shuffled*) this autocorrelation is broken by randomisation over the instances order.

Figure 6 plots the accuracy results of the compared approaches on three versions of the Cover Type dataset: the original, sorted (gradual drifts), and shuffled (no drift).

We can observe that all the algorithms obtain similar performances on the original data (Figure 6(a)) while on the sorted version (Figure 6(b)) we can note that DBALStream obtains a slight improvement w.r.t. the competitors for budget values around 0.1.

Considering the *Cover Type Shuffled* version (Figure 6(c)), DBALStream clearly outperforms all the competitors for all the budget values. The only exception happens for

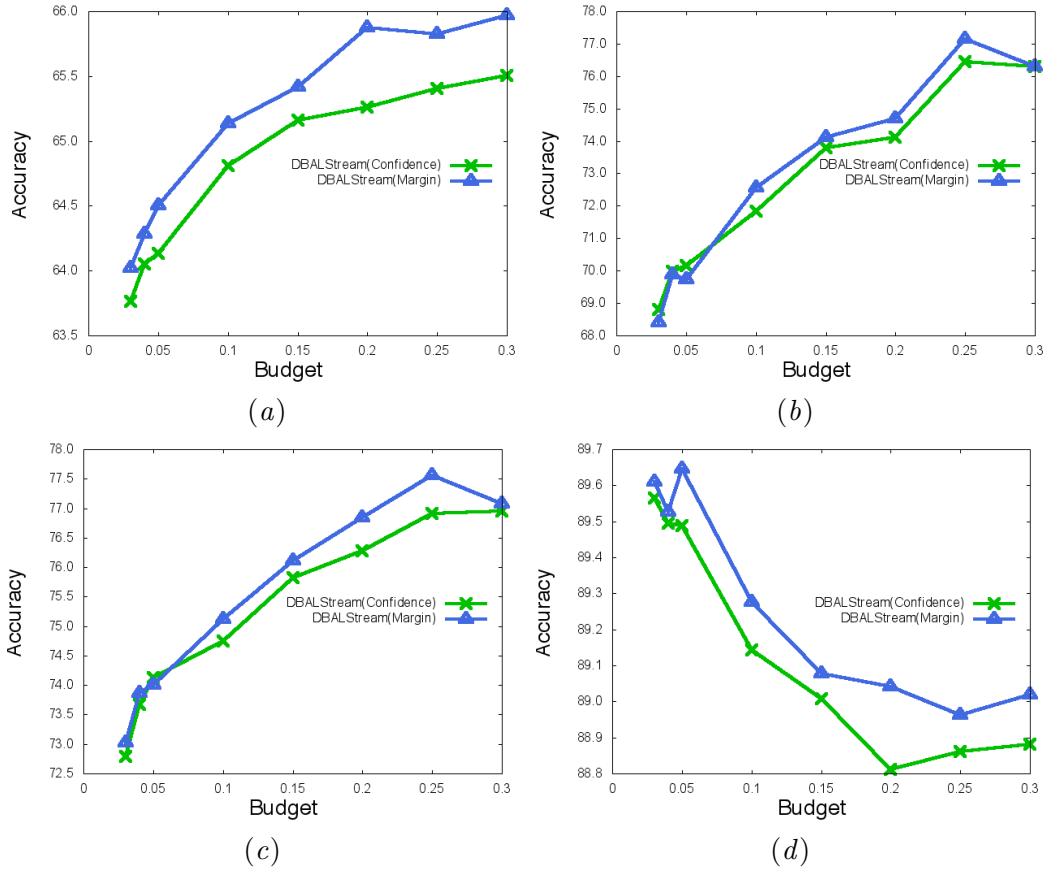

Figure 4: Accuracy Results for DBALSTREAM coupled with different *Confidence* based and *Margin* based uncertainty estimation on a) *Airlines* b) *Cover Type* c) *Electricity* and d) *KDD99*

budget value equals to 0.05 where the *Rand Unc* heuristic obtains the same results as our proposal. This experiment underlines that DBALSTREAM performs similarly to state of the art approaches for spatially auto-correlated data while it clearly outperforms the competitors when data instances are not affected by this kind of auto-correlation.

5. Conclusions

Building classification models on data streams considering only a limited amount of labeled data is becoming a common task due to time and cost constraints.

In this paper we presented DBALSTREAM, a novel algorithm to perform active learning in a data stream scenario. Our approach exploits local instance correlation over the feature space in order to improve sampling on the potentially most useful examples to label.

We assessed the performance of our proposal over real world benchmark datasets. The results showed that the proposed approach outperforms state-of-the-art active learning strategies for data streams in terms of predictive accuracy. We empirically studied dif-

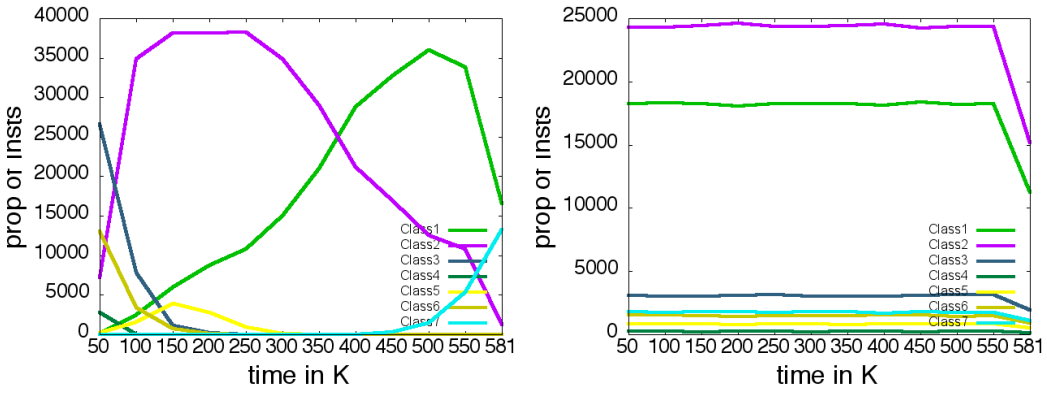

Figure 5: Distribution of the class values during the stream process: a) *Cover Type Sorted* b) *Cover Type Shuffled*

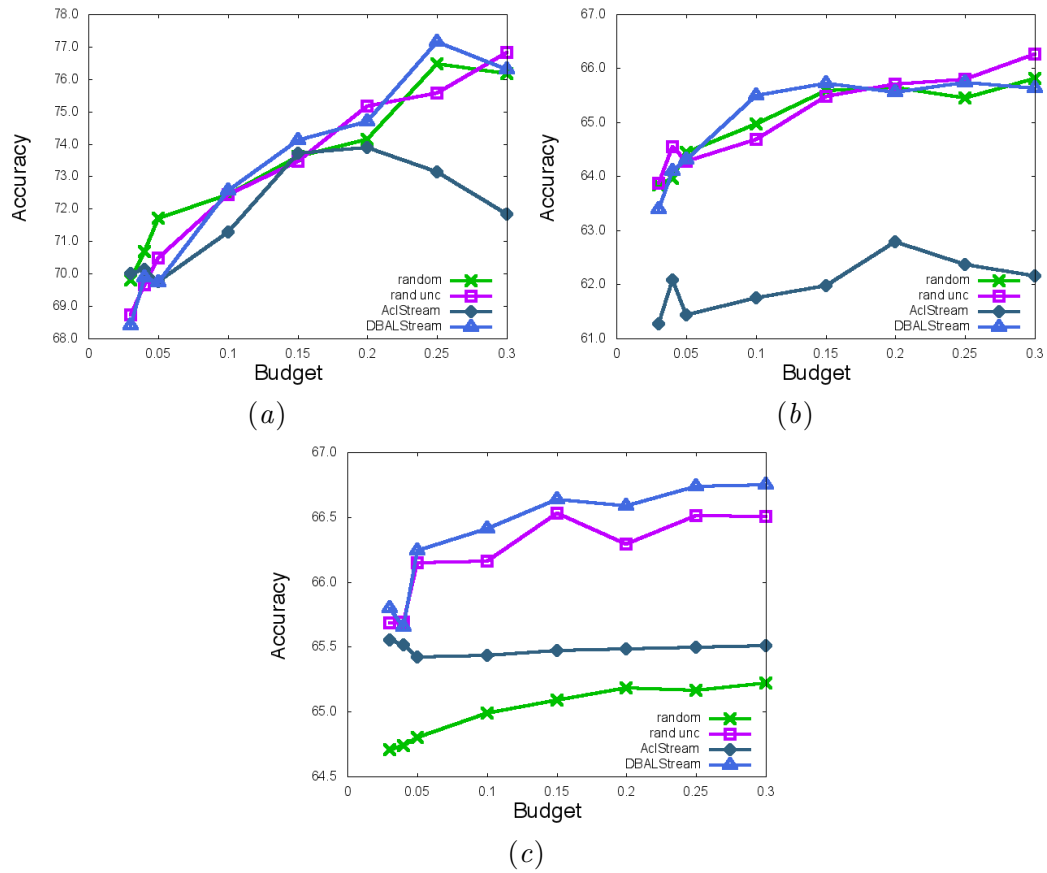

Figure 6: Accuracy on a) *Cover Type* b) *Cover Type Sorted* and c) *Cover Type Shuffled*

ferent factors that can impact our strategy: the window size used to estimate the local density of a new instance and the way in which the uncertainty of an example is estimated. As a final experiment we supplied an in-depth study on how our strategies deal with evolving data and manage concept drift.

As future work we would like to investigate other ways to measure and incorporate density into the active learning process for data streams.

References

J. Attenberg and F. Provost. Online active inference and learning. In *Proc. of the 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD, pages 186–194, 2011.

K. Bache and M. Lichman. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2013.

A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, 11(May):1601–1604, 2010.

N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Worst-case analysis of selective sampling for linear classification. *J. Mach. Learn. Res.*, 7:1205–1230, 2006.

D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15:201–221, 1994.

W. Fan, Y. Huang, H. Wang, and Ph. Yu. Active mining of data streams. In *Proc. of SIAM Int. Conf. on Data Mining*, SDM, pages 457–461, 2004.

Y. Fu, X. Zhu, and B. Li. A survey on instance selection for active learning. *Knowl. Inf. Syst.*, 35(2):249–283, 2013.

J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Proc. of the 17th Brazilian Symp. on Artificial Intelligence*, SBIA, pages 286–295, 2004.

M. Harries, C. Sammut, and K. Horn. Extracting hidden context. *Machine Learning*, 32 (2):101–126, 1998.

D. Helmbold and S. Panizza. Some label efficient learning results. In *Proc. of the 10th Annual Conf. on Computational Learning Theory*, COLT, pages 218–230, 1997.

Sh. Huang and Y. Dong. An active learning system for mining time-changing data streams. *Intelligent Data Analysis*, 11:401–419, 2007.

D. Ienco, A. Bifet, I. Zliobaite, and B. Pfahringer. Clustering based active learning for evolving data streams. In *Proc. of the 16th Int. Conf. on Discovery Science*, DS, pages 79–93, 2013.

E. Ikonomovska, J. Gama, and S. Dzeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, 2010.

R. Klinkenberg. Using labeled and unlabeled data to learn drifting concepts. In *Proc. of IJCAI workshop on Learning from Temporal and Spatial Data*, pages 16–24, 2001.

D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proc. of the 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, SIGIR, pages 3–12, 1994.

P. Lindstrom, S. J. Delany, and B. MacNamee. Handling concept drift in a text data stream constrained by high labelling cost. In *Proc. of the 23rd Florida Artificial Intelligence Research Society Conference*, FLAIRS, 2010.

M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham. Classification and novel class detection in data streams with active mining. In *Proc. of the 14th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining*, PAKDD, pages 311–324, 2010.

M. Masud, C. Woolam, J. Gao, L. Khan, J. Han, K. Hamlen, and N. Oza. Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowl. Inf. Syst.*, 33 (1):213–244, 2011.

A. Rodriguez and A. Laio. Clustering by fast search and find of density peaks. *Science*, 344 (6191):1492–1496, 2014.

D. Sculley. Online active learning methods for fast label-efficient spam filtering. In *Proc. of the 4th Conf. on Email and Anti-Spam*, CEAS, 2007.

B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison, 2010.

N. Tomasev, M. Radovanovic, D. Mladenic, and M. Ivanovic. The role of hubness in clustering high-dimensional data. *IEEE Trans. Knowl. Data Eng.*, 26(3):739–751, 2014.

D. Widyantoro and J. Yen. Relevant data expansion for learning concept drift from sparsely labeled data. *IEEE Trans. on Know. and Data Eng.*, 17:401–412, 2005.

P. Zhao and S. C. H. Hoi. Cost-sensitive online active learning with application to malicious URL detection. In *Proc. of the 19th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD, pages 919–927, 2013.

X. Zhu, P. Zhang, X. Lin, and Y. Shi. Active learning from data streams. In *Proc. of the 7th IEEE Int. Conf. on Data Mining*, ICDM, pages 757–762, 2007.

I. Zliobaite, A. Bifet, B. Pfahringer, and G. Holmes. Active learning with drifting streaming data. *IEEE Trans. on Neural Networks and Learning Systems*, 25:27–39, 2014.